Tosta-hub

- Grupo 1 y 3
- Curso escolar: 2024/2025
- Asignatura: Evolución y gestión de la configuración

Miembro	Implicación
Aguayo Orozco,Sergio	10
Dana Cabello,David	10
García Parras,Luis	10
Mateos Angulo,Pablo	10
Naredo Bernardos,Ignacio	10

Enlaces de interés

Sistema de despliegue: https://tostahub-dev.onrender.com/

Despliegue en servidor privado: https://egc.imalittle.baby

User: invit4de, **password**: invit4de

Esta autenticación está puesta en la configuración del server para evitar que nadie que obtenga la url pueda hacer nada negativo. No es inicio de sesión de TostaHub en sí.

Repositorio Github: https://github.com/TostaHub/tostahub-uvlhub.git

Miembro del equipo	Horas	Commits	Loc	Test	Issues	Work Item
Aguayo Orozco, Sergio	60	61	2053	12	17	Advance filter
Dana Cabello, David	60	16	727	18	4	Advance query search
García Parras, Luis	60	19	719	7	6	Edit dataset metadata
Mateos Angulo, Pablo	60	37	1347	15	8	Download in different formats, view user profile
Naredo Bernardos, Ignacio	60	36	2420	8	10	Rate datasets, Fakenodo

Resumen ejecutivo

En el proyecto “TostaHub” abordamos la tarea de realizar modificaciones en el proyecto proporcionado como base “UvlHub”, con este objetivo en primer lugar establecimos como work items los cambios que queríamos realizar, estos fueron: **edit dataset**, **rate dataset**, **advanced filter**, **view dataset**, **view user profile** y **download in different formats** y **advanced search query**. Además de estos también realizamos el work item fake nodo, con el objetivo de no colapsar de peticiones la API Zenodo, permitiendo que las pruebas del proyecto se realizarán sin interrupciones.

El objetivo de esta asignatura es comprender que para que un proyecto software profesional salga adelante y cumpla los requisitos mínimos exigibles desde la profesionalidad es necesario que, como equipo, sigamos unas mismas normas y una forma de trabajar unificada. Con ese objetivo, en primer lugar, establecimos una serie de normas a seguir por todos los miembros del equipo. Estas normas incluyen estrictas políticas de commits, ramas y issues. Estas normas son muy importantes en un proyecto software profesional, ya que contribuyen a que el trabajo sea

homogéneo, organizado y transparente. Además, facilitan que tanto los miembros actuales del equipo como posibles futuras incorporaciones puedan entender el trabajo de los demás miembros de manera sencilla y eficiente.

Además de los documentos de planificación, como en cualquier proyecto software profesional, elaboramos un acta fundacional. En este documento describimos los posibles conflictos que se pueden producir durante un proyecto software, como desavenencias en la toma de decisiones o problemas de comunicación, y detallamos cómo queríamos abordarlos en caso de que se produjeran. El acta nos permitió tener una guía clara de resolución de problemas, evitando bloqueos en el desarrollo.

Asimismo, también elaboramos un diario de equipo. En este diario quedaron registradas todas las reuniones que hemos mantenido como equipo, especificando los participantes, las fechas en las que se llevaron a cabo y los objetivos que surgieron de cada reunión. Gracias a este diario, conseguimos tener un historial detallado y transparente del proceso que ha seguido el proyecto software desde su inicio hasta su desarrollo actual.

Para controlar y garantizar que el equipo cumpliese las políticas mencionadas anteriormente, se establecieron una serie de restricciones utilizando las herramientas que GitHub nos proporciona. Así, se creó una rama de producción llamada Develop que está protegida y solo puede ser modificada con la aprobación de los cambios por parte de un miembro diferente al que haya realizado las modificaciones. Esto garantiza que todo cambio introducido en el proyecto haya sido revisado y validado por al menos dos miembros del equipo, mejorando considerablemente la calidad del código y reduciendo la posibilidad de errores.

Además, para asegurar que se cumplan las buenas prácticas de programación y las restricciones propias de un proyecto software profesional, también se implementaron una serie de workflows automáticos. Estos workflows se ejecutan en cada pull request realizada, realizando tests automáticos que comprueban la integridad del código. De esta forma, se garantiza que los cambios introducidos no rompan ninguna funcionalidad existente y que cumplan con los estándares establecidos previamente. La combinación de las reviews de código y los tests automáticos nos permitió tener una rama estable y segura, reduciendo considerablemente muchas de las incidencias típicas que surgen en proyectos software.

Por otro lado, en el desarrollo de las nuevas funcionalidades, nos enfocamos en asegurar que fueran intuitivas, eficientes y de fácil uso. Por ejemplo, en la funcionalidad rate dataset, permitimos a los usuarios puntuar datasets de forma sencilla a través de una interfaz gráfica amigable. Esta funcionalidad se implementa de manera que las calificaciones sean almacenadas correctamente y se puedan

visualizar de forma agregada, permitiendo que otros usuarios puedan evaluar la calidad de un dataset antes de descargarlo.

Otra funcionalidad clave fue la implementación del advanced filter, que permite a los usuarios filtrar datasets de acuerdo a diferentes criterios como fecha, autor, categoría y puntuación. Esta funcionalidad facilita la navegación y mejora considerablemente la experiencia del usuario al reducir el tiempo de búsqueda de datasets relevantes.

Finalmente, también implementamos la posibilidad de" download in different format", una funcionalidad que ofrece a los usuarios la opción de descargar los datasets en formatos como CSV, JSON y XML, ampliando las posibilidades de uso y adaptabilidad a diferentes contextos y necesidades profesionales.

Gracias a la correcta organización del equipo, el uso de herramientas profesionales como GitHub y la implementación de políticas estrictas de trabajo, logramos que el proyecto "TostaHub" alcanzará los objetivos planteados inicialmente. Este proceso no solo nos permitió desarrollar un software funcional y de calidad, sino también adquirir experiencia en el trabajo en equipo y en la aplicación de buenas prácticas que son indispensables en cualquier proyecto software profesional.

Descripción del sistema

El sistema implementado recibe el nombre de UvlHub, es un sistema creado por la universidad de Sevilla, la universidad de Málaga y la universidad de Ulm en Alemania.El objetivo de este sistema es proporcionar a los investigadores un sistema web en que dar soporte a sus datasets.

Un dataset es una colección estructurada de datos relacionados que se organiza de manera que sea accesible, gestionable y utilizable para su análisis o procesamiento. Generalmente, un dataset contiene registros individuales organizados en filas y columnas, para nuestro proyecto hemos intentado ampliar el proyecto base de Uvlhub, añadiendo a este una serie de funcionalidades que creemos pueden ayudar bastante tanto a quienes quieran publicar en UvlHub como quienes quieran consultarlos.

Con este objetivo hemos añadido un sistema de filtrado complejo que debería facilitar a quien consulte la página la búsqueda de aquellos datasets que sean de su interés, este advance filter incluye la posibilidad de búsqueda por la fecha en la que el dataset ha sido creado, la búsqueda por un mínimo y máximo de UVLs contenidos dentro del dataset y la búsqueda estableciendo un mínimo y un máximo de configuraciones, además, por último se ha añadido la posibilidad de ordenarlo empezando por los más nuevos o por los más antiguos.

La segunda funcionalidad que hemos añadido es search query, con esta funcionalidad pretendemos que además de la posibilidad de filtrar los datasets el usuario pueda también buscar directamente un dataset concreto, creemos que esta funcionalidad es de gran utilidad puesto que facilita a los usuarios buscar de forma rápida y sencilla aquellos datasets que le parezcan de su interés

La tercera funcionalidad implementada por nuestro equipo de trabajo es la de rate dataset, con esta funcionalidad implementada cada dataset puede ser evaluado del 1 al 5, con esta funcionalidad pretendemos que los usuarios puedan distinguir perfectamente y a simple vista si un dataset es considerado de calidad o no por otros usuarios, consiguiendo así que aquellos dataset de calidad están bien valorados y se vean recompensados y filtrar aquellos que no sean de buena calidad, bien para que el creador pueda mejorarlos, bien para que no hagan perder tiempo a los investigadores que deseen consultar datasets relacionados con su campo de investigación, además con esta funcionalidad creemos que se fomenta que la aplicación web se convierta en un entorno colaborativo en los que distintos usuarios colaboren de manera conjunta para ayudarse los unos a los otros.

Otra de las funcionalidades que han sido implementadas por nuestro equipo de desarrollo es la implementación de la posibilidad por parte de los creadores de datasets de editar sus datasets ya subidos, esta funcionalidad nos permite extremadamente útil ya que ahorra el trabajo al creador de tener que borrar y subir de nuevo el dataset cuando el cambio sea de pequeña importancia.

Las otras dos implementaciones que hemos realizado como proyecto son la posibilidad de ofrecer una vista tanto del perfil como del dataset, de esta manera se consigue que la información relevante tanto de los datasets como del perfil esté presente en el proyecto de manera clara y ordenada.

La última de las funcionalidades implementadas por nuestro equipo de trabajo se trata del fake nodo, esta funcionalidad es de extrema utilidad por muchos aspectos que describiremos a continuación. en primer lugar, la sustitución de Zenodo nos asegura que no colapsaremos su sistema mientras probamos las distintas funcionalidades, ya que al ser sustituido por este fake nodo nos aseguramos de que todas las pruebas vayan a nuestro entorno local,como segunda funcionalidad principal nos aseguramos de que podamos probar todo el funcionamiento sin conexión a internet, algo que puede ser de extrema utilidad en casos en los que no se pueda tener acceso a internet.

Por último para probar que todas las funcionalidades funcionen de manera correcta y no haya ningún comportamiento no esperado se han realizado pruebas de todas estas funcionalidades, estas pruebas han sido de tres tipos, de tipo unitario, de carga y de tipo gráfico.

Pruebas unitarias: Con el objetivo de verificar y controlar el correcto comportamiento de todas las funcionalidades implementadas hemos creado pruebas unitarias que cumplan correcto funcionamiento de las funcionalidades, que ocurre en caso de incorrecto funcionamiento y qué ocurre si el usuario intenta realizar algo que o no está permitido o no es posible realizar en el proyecto software.

Pruebas de carga: Con el objetivo de comprobar que el sistema sea capaz de soportar toda la carga de trabajo requerida por nuestras nuevas funcionalidades hemos implementado, usando locust, pruebas de carga que simulen el comportamiento de la web, esto nos ha parecido extremadamente útil puesto que locust permite implementar tanto el número de usuarios a simular como el número de peticiones a implementar.

Por último hemos implementado pruebas de interfaz gráfica usando selenium, esta herramienta es extremadamente útil ya que además de asegurar el correcto funcionamiento de la página nos permite verlo de manera gráfica, lo cuál facilita la asimilación de lo que se está haciendo, además estas pruebas son muy fáciles de implementar pues permiten realizarlas mediante una extensión en el próximo navegador, esta extensión nos permite grabar directamente sobre el mismo navegador la funcionalidad que queremos que el test simula, de esta manera estas pruebas se vuelven extremadamente intuitivas y fáciles de comprender.

Este sistema ha sido desplegado en local para su prueba y realización, así mismo ha sido desplegado en Render, y en un servidor privado de uno de los miembros de equipo,ambos links se encuentran en la portada del presente documento, de esta manera hemos conseguido que el sistema esté disponible tanto en render como en un servidor de nuestra propiedad,

Por último queremos describir los workflows implementados en el proyecto con el objetivo de dotarlo de profesionalidad, estos workflows han añadido dos test tanto unitarios como de lint en la rama de producción, de nombre develop, los cuales garantizan que los cambios realizados por cualquier miembro del equipo cumplan los requisitos mínimos establecidos por el equipo.

También en la rama principal, de nombre main, además de estos dos workflows se han implementado workflows extras que nos aseguran aún más la estabilidad de esta rama, la cuál es más sensible pues se trata de la principal.

Por último destacar que en cualquier pull request solicitada por cualquiera de los miembros del equipo se comprueba automáticamente que el despliegue a render se realice de manera correcta, de esta manera nos aseguramos que ninguna de las versiones producidas por nuestro equipo no funcione.

Visión global del proceso de desarrollo

El proceso de desarrollo de UvlHub, un proyecto basado en Flask para la gestión y visualización de datasets, se ha caracterizado por una meticulosa planificación, organización y ejecución, con el objetivo de proporcionar una plataforma robusta y funcional para la comunidad científica y profesional. El proyecto ha pasado por diversas etapas, desde la concepción de la idea y la creación de su estructura inicial hasta la implementación de nuevas funcionalidades y la optimización de su rendimiento, todo ello utilizando herramientas profesionales y buenas prácticas de desarrollo de software.

En este contexto, se puede identificar una serie de fases y actividades clave que han guiado el progreso de UvlHub, permitiendo que el proyecto se mantuviera alineado con los objetivos iniciales, cumpliendo con los estándares de calidad y asegurando su escalabilidad y mantenimiento en el futuro.

Como objetivo inicial se planteó definir claramente cuáles eran los objetivos que se pretendían conseguir con la realización de este proyecto, con esto en mente definimos claramente cuáles eran los work items que deseábamos implementar, estos work items fueron aprobados por el tutor de nuestro proyecto en el M1, como segunda fase nos decidimos a llevar a cabo la implementación en sí de estas funciones, para el desarrollo de estas funciones se estableció una serie de políticas concretas y exactas.

Para la política de commits se decidió seguir una política estricta que consiste en que cada uno de ellos tienen que empezar con unos de los verbos establecidos en mayúscula seguido de dos puntos, para a continuación seguir con otro verbo y una pequeña descripción de lo que consiste el commits en sí.

Para la política de ramas se decidió seguir también una estricta política, esta consistía en la creación de dos ramas principales del trabajo, la rama main de producción y la rama develop de desarrollo. En la rama develop se irían implementando cada uno de los cambios y una vez comprobado su correcto funcionamiento estos se pasarían a main, además de estas ramas también se decidió que existirán ramas auxiliares para la creación de las funcionalidades.

El proceso en sí de cada cambio, consiste en una vez decidida la funcionalidad que se quiere implementar acudir a Github y crear en el proyecto una nueva issue que llevará como título el nombre del work item que se quiere realizar, en caso de que no sea una funcionalidad en si se le pondrá la palabra que mejor lo describa delante, por ejemplo, fix si se trata de un arreglo o test si lo que se está haciendo son pruebas.

Una vez realizado el issue se le deberá asignar a este un encargado de su implementación así como una prioridad, que puede ser desde cero hasta tres, también se le debe asignar una de las etiquetas disponibles en el proyecto.

Una vez creada la issue cuando se decida comenzar a trabajar en esta funcionalidad en concreto se moverá la issue del panel to do al panel in progress, para comenzar a trabajar en ella se usará el botón create branch disponible en Github, el cual creará una branch con el nombre de la issue, esto nos permite seguir con claridad qué issue pertenece a cada rama, el encargado del equipo de desarrollo de esta tarea deberá cambiar a esa rama en su entorno local, usando los comandos apropiados para ello, y ponerse a trabajar en implementar los cambios

A continuación, se realizará el commit siguiendo las políticas previamente indicadas, una vez el commit este realizado y la funcionalidad a implementar decidida el encargado de esa parte del proyecto decidirá abrir la pull request a la rama develop, una vez comprobado que la pull request pasaba los test estipulados para cada pull request, el integrante del equipo encargado de esa funcionalidad tendrá que pedirle a algún otro miembro del equipo que le revise los cambios y o bien se los apruebe en caso de que los considere correcto, o por el contrario se los deniegue si cree que los cambios no son los adecuados, en caso de que los cambios hayan sido aceptados por el otro miembro del equipo, el encargado original de implementar esta funcionalidad podrá realizar el merge a la rama de desarrollo y una vez se haya comprobado que esta rama está estable se realizará el siguiente merge a main.

En caso de que los cambios no hayan sido satisfactorios el miembro del equipo que actúa como revisor tendrá que rechazar la pull request y comentar lo que , en su opinión, el cambio necesita para ser aceptado, el miembro encargado del cambio leerá los comentarios y actuará en consecuencia, a continuación de cambiar su funcionalidad en base a los comentarios del otro miembro del equipo podrá volver a realizar la pull request y el proceso de revisión comenzará de nuevo, si los cambios son esta vez sí son aceptados podrá realizar el merge a develop comprobar el correcto funcionamiento de esta rama tanto en local como en render y finalmente integrar en la rama de desarrollo para que los cambios queden aceptados en el proyecto.

Una vez este cambio se ha producido correctamente se procede a borrar la rama creada únicamente para realizar esa issue, de esta manera se consigue que el GItHub del equipo tenga una apariencia profesional, el objetivo por tanto es que una vez finalizado el trabajo solo existan, de nuevo, las dos ramas principales estipuladas en el inicio del proyecto para conseguir que el repositorio del equipo tenga la apariencia más profesional posible.

El objetivo de este proceso de desarrollo es otorgar a nuestro proyecto de las características propias de un proyecto profesional, con el objetivo de que todo el

equipo se familiarice con lo que sería la participación en un proyecto de la vida real, para ellos hemos implementado las políticas propias de una empresa real, así como sus controles de calidad y de funcionalidad. El otro objetivo de este proceso de desarrollo es asegurarnos de reducir al máximo posible los riesgos propios de un proyecto software, por eso se han implementado workflows de control y la necesidad de aprobar todos los cambios que se produzcan en el proyecto, de esta manera hemos conseguido realizar el proyecto sin graves incidencias, con claridad y de manera eficiente.

Entorno de desarrollo

El sistema operativo usado para la realización de nuestro proyecto software ha sido **Ubuntu 22.04 LTS**. Como IDE los miembros del equipo realizamos reunión para discutirlo, fruto de esa reunión surgió la decisión consensuada de usar **VSCode** además con el objetivo de facilitar la tarea de implementación de nuestras funcionalidades y de la compresión del código base decidimos que todos los miembros del equipo instalarán las siguientes extensiones disponibles en visual studio code: : **Docker, Python, Debugpy, Python indent, Flake8, ESLint, PyPi Assistant, Flask Snippets**. Como base de datos para el proyecto se decidió, como equipo, usar MariaDB debido a su facilidad de uso y su compatibilidad con el sistema operativo elegido.

El proyecto está hecho usando el framework Flask, este framework está escrito en python por lo que es necesario instalar este lenguaje de programación, en concreto para nuestro trabajo hemos decidido usar la versión 3.12 ya que es la que mejor se adapta a nuestras necesidades y menos problemas de compatibilidad dar.

Para tener el entorno de desarrollo preparado se siguen los siguientes pasos, los cuales fueron seguidos por todo el equipo de desarrollo:

Clonar proyecto: **git clone git@github.com:TostaHub/tostahub-uvlhub.git**

Entrar en la carpeta creada: **cd tostahub-uvlhub**

Añadir repositorio necesario para python: **sudo add-apt-repository ppa:deadsnakes/ppa**

Actualizar lista de paquetes: **sudo apt update**

Instalar python: **sudo apt install python3.12**

Instalar MariaDB: **sudo apt install mariadb-server -y**

Iniciar servicio: **sudo systemctl start mariadb**

Configuración inicial: **sudo mysql_secure_installation**

- **Enter current password for root (enter for none): (enter)**
- **Switch to unix_socket authentication [Y/n]: `y`**
- **Change the root password? [Y/n]: `y`**
  - **New password: `uvlhubdb_root_password`**
  - **Re-enter new password: `uvlhubdb_root_password`**
- **Remove anonymous users? [Y/n]: `y`**
- **Disallow root login remotely? [Y/n]: `y`**
- **Remove test database and access to it? [Y/n]: `y`**
- **Reload privilege tables now? [Y/n] : `y`**

Entrar en la base de datos: **sudo mysql -u root -p** con `uvlhubdb_root_password` como contraseña para crear lo siguiente:

```
CREATE DATABASE uvlhubdb;

CREATE DATABASE uvlhubdb_test;

CREATE USER 'uvlhubdb_user'@'localhost' IDENTIFIED BY 'uvlhubdb_password';

GRANT ALL PRIVILEGES ON uvlhubdb.* TO 'uvlhubdb_user'@'localhost';

GRANT ALL PRIVILEGES ON uvlhubdb_test.* TO 'uvlhubdb_user'@'localhost';

FLUSH PRIVILEGES;

EXIT;
```

Copiar variables de entorno base: **cp .env.local.example .env**

Añadir webhook a .moduleignore: **echo "webhook" > .moduleignore**

Instalar python venv: **sudo apt install python3.12-venv**

Crear entorno python: **python3.12 -m venv venv**

Activar entorno: **source venv/bin/activate**

Actualizar pip: **pip install --upgrade pip**

Instalar requisitos python: **pip install -r requirements.txt**

Necesario para rosemary: **pip install -e ./**

Migraciones: **flask db upgrade**

Poblar tablas: **rosemary db:seed**Para correr la aplicación: **flask run --host=0.0.0.0 --reload --debug**

Una vez finalizada esta configuración todos los miembros de nuestro proyecto estaban preparados para comenzar a realizar los objetivos marcados por el proyecto.

Ejercicio de propuesta de cambio

La propuesta de cambio es cambiar el nombre de las etiquetas “Filter number of uvl models” y “Filter number of configurations” a “Filter by number of uvl models” y “Filter by number of configurations” respectivamente.

Para ello se creará un nuevo issue en la página https://github.com/TostaHub/tostahub-uvlhub/issues/new/choose, en concreto se creará un bug report. Se rellenará la descripción, los pasos a seguir y el comportamiento esperado. A este bug report se le dará una prioridad, será asignado a un compañero y se creará una rama para su desarrollo. Este compañero llevará a cabo los cambios en el código, lo cual será cambiar esas dos etiquetas en app/modules/explore/templates/explore/index.html. Hará **git fetch origin**, **git checkout <nuevaRramaCreada>**, cambios, **git add .** (No debería haber ningún otro archivo modificado), **git commit -m “fix: Add a more expressive message in filter tags”**, **git push**. Una vez subido los cambios otro compañero creará la pull request a **develop** (esto normalmente lo hace el mismo que trabaja en la rama, pero se pide que cada compañero participe en el flow en este ejercicio). Otro compañero revisará los cambios y los aceptará y el último compañero hará merge (normalmente se espera a que el despliegue termine) y borrará la rama.

Esto hará que los workflows siguientes se ejecuten:

- **test.yml**: Se ejecutan los tests.
- **lint.yml**: Se comprueba que el formato del código sea correcto.
- **render.yml**: Se despliega develop en render y se monitoriza que sea exitoso.

Luego se hará una pull request a main, aceptará y mergeará para que los siguientes workflows se ejecuten:

- **test.yml**: Se ejecutan los tests.
- **lint.yml**: Se comprueba que el formato del código sea correcto.
- **render.yml**: Se despliega main en el servidor personal. No se monitoriza porque el despliegue es más rápido y devuelve su estado inmediatamente.

Conclusiones y trabajo futuro

Como conclusión del trabajo queremos destacar que esta asignatura nos ha parecido de gran utilidad fundamentalmente en el acercamiento como alumnos de ingeniería del software que están apunto de finalizar sus estudios a cómo sería realmente un proyecto software en la vida real, nos parece muy destacable que esta asignatura no se centre tanto en el código, puesto que esto ya se ha visto sobradamente durante todo el grado, sino que se centre más en los aspectos de la ingeniería, es decir, que se centre en que un proyecto para ser considerado profesional y de ingeniería tiene que cumplir una serie de normas y de buenas prácticas que aseguren su éxito. Nos ha sido muy útil para familiarizarnos a fondo con todas las herramientas que ofrecen tanto Github como Git, y para darnos cuenta de que estas herramientas aunque puedan parecer complicadas en un principio, una vez entendidas y usadas se convierten en una ayuda enorme para el trabajo en equipo, el control del proyecto y su finalización con éxito.

Como mejoras de cara a años posteriores nos gustaría proponer la posibilidad de añadir un botón que permitiese compartir un dataset por otros servicio tales como, correo electrónico, o alguna red social, otra propuesta que nos parece interesante es la posibilidad de implementar la posibilidad de crear una pantalla de información sobre los autores de los datasets, en esa pantalla podrían aparecer elementos de información sobre ellos tales como un correo de contacto, un teléfono o un enlace a redes sociales de ámbito profesional como podría ser Linkedin.